(12) UK Patent Application (19) GB (11) 2 355 319 (13) A

(43) Date of A Publication 18.04.2001

(21) Application No 9924224.0

(22) Date of Filing 14.10.1999

(71) Applicant(s)
International Business Machines Corporation
(Incorporated in USA - New York)
Armonk, New York 10504, United States of America

(72) Inventor(s)
Mark Ansell

(74) Agent and/or Address for Service
C Boyce
IBM United Kingdom Limited, Intellectual Property Dept, Hursley Park, WINCHESTER, Hampshire, SO21 2JN, United Kingdom

(51) INT CL^7
G06F 9/48

(52) UK CL (Edition S)
G4A AFN

(56) Documents Cited
None

(58) Field of Search
UK CL (Edition R) G4A AFN
INT CL^7 G06F
ONLINE:WPI,EPODOC,JAPIO

(54) Abstract Title
Job scheduler

(57) A Java job scheduler 10 operable in a Java virtual machine 20 is disclosed. The scheduler has an associated task file 40 comprising one or more platform dependent task definitions having associated conditions for execution. The scheduler comprises a thread 12 for monitoring for a change to the task file; and a thread 14, responsive to changes in the task file, for reading the task file and, responsive to the conditions for execution of any one of the one or more tasks being met, spawning a further thread 16 for executing the task.

The invention can be implemented in a platform independent language other than Java, e.g. PERL.

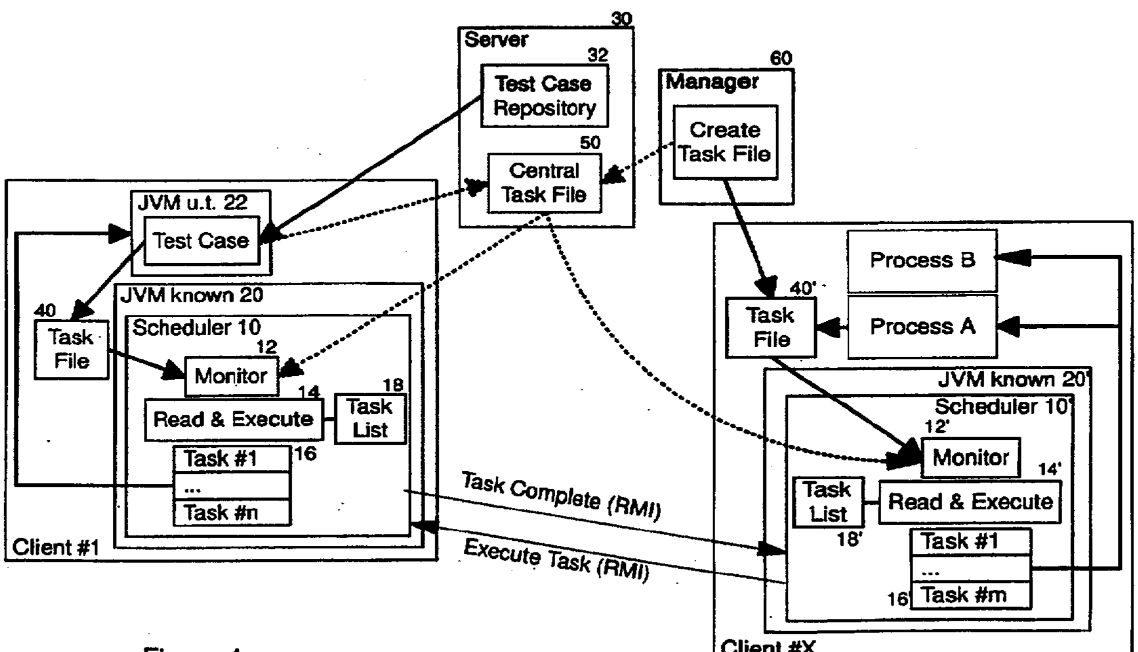

Figure 1

GB 2 355 319 A

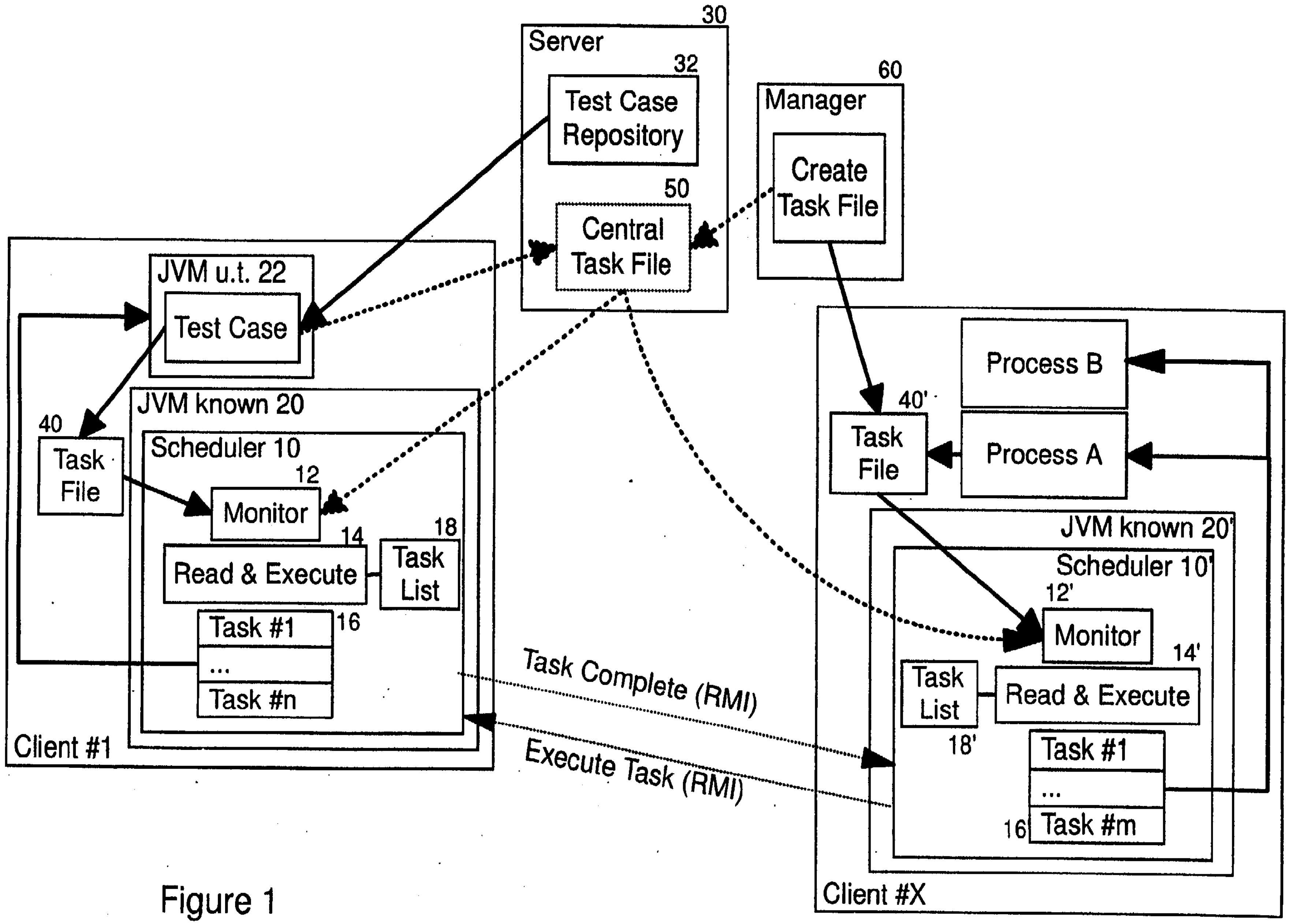
Server
30
Test Case Repository
32
Central Task File
50
Manager
60
Create Task File
Client #1
JVM u.t. 22
Test Case
JVM known 20
Scheduler 10
Task File
40
Monitor
12
Read & Execute
14
Task List
18
Task #1
...
Task #n
16
Task Complete (RMI)
Execute Task (RMI)
Client #X
Process B
Process A
Task File
40'
JVM known 20'
Scheduler 10'
Monitor
12'
Read & Execute
14'
Task List
18'
Task #1
...
Task #m
16

Figure 1

```
# Jsched.txt
#
1 07 12 * *  6  * job1.out java -version
2 53 14 * * 0-6 * job2.out ksh -x /jck/newautojck/java/autojckma.sh
3 * * * * * 2 job3.out d:\jck\newautojck\before.bat
4 * * * * * 1 job4.out java -version
5 * * * * * 2 job5.out d:\jck\newautojck\after.bat
6 * * * * * 4 job4.out java -version
#
```

Figure 2

JOB SCHEDULER

The present invention relates to a job scheduler and in particular a job scheduler written in a platform independent language such as Java, allowing for deployment of the scheduler in a distributed heterogeneous environment.

Conventional scheduler utilities, for example, Cron supplied with the AIX operating system produced by IBM Corporation are cooperable with a task file which includes a list of tasks to be executed and the date/time or frequency for executing those tasks. (The terms job and task are used interchangeably in the present specification.) Although awkward, the task file can be written to define lists of tasks which execute sequentially as well as tasks which execute independently of one another. To alter the schedule, the task file must be edited and the scheduler explicitly instructed to re-read the task file. This can involve problems with tracking the state of tasks, particularly if the task file is edited during task execution and the effect this may have on dependent sequential tasks.

The use of scheduler utilities is becoming more and more common, particularly by developers who may need to repetitively run complex tasks. Such developers are often faced with the prospect of needing to run such complex tasks across distributed heterogeneous platforms and end up using platform specific schedulers for each platform on which they are working. This of course entails learning how to configure each scheduler, but more importantly it makes it extremely difficult to schedule tasks running on one machine, possibly on a first platform, with dependent tasks running on other machines, possibly running on different platforms.

Accordingly, the present invention provides a job scheduler as claimed in claim 1.

As will be described, the invention enables, for example, a developer to test releases of Java across multiple platforms. In this case, the scheduler automatically runs the tests once a new release/build becomes available, normally each morning following an overnight build process. Thus, using the invention, the developer is able to define and run tests as required on all Java-supported platforms.

The scheduler can use a similar task file input to prior art schedulers such as Cron to define what is to run and when. The advantage is, being written in a platform independent language, such as Java, the scheduler can be used as a general purpose tool across many platforms. It does not have to be used to run Java programs - any batch file or script containing platform specific commands or executable can be called.

It should also be seen that Java and similar languages have been commonly used as a quick way to implement applications which are of course platform independent. On the other hand, because the functionality of operating system utilities is usually tied to an operating system or platform, utilities are normally compiled for direct execution on a platform. By contrast, the scheduler according to the invention is a portable utility.

Embodiments of the invention will now be described with reference to the accompanying drawings, in which:

Figure 1 is a schematic diagram of a distributed heterogeneous system including the job scheduler according to the invention; and

Figure 2 is a sample task file for use with the job scheduler of Figure 1.

Referring now to Figure 1 the scheduler 10 comprises a class instantiated by a Java virtual machine (JVM) 20. Where the platform supports Java JDK 1.1.6 or later, the class may be stored in a Java Class Archive file (JAR) resident on the same machine as the JVM 20 or even on a remote machine accessible to the JVM.

In the case of client machine #1, the scheduler 10 is used to execute tasks for testing a new release of JVM. This JVM under test 22 is built and a number of test cases comprising Java applications are run on the JVM 22 to assess its functionality. Typically, the test cases are stored in a central server 30 within a repository 32, making the cases available for execution on JVMs to be tested across a number of platforms.

In operation the scheduler 10 includes two threads: a monitor thread 12 and a read and execute thread 14. The monitor thread 12 continually monitors a task file 40 to determine when the task file changes. This can occur: when a user edits the file 40 and saves it as a new version; when a new version is sent from a remote machine (via FTP for example); or when some other process running on either the local or remote machine updates the task file either by replacing the file or editing the file.

An example task file is shown in Figure 2. In the preferred embodiment, each task occupies a single line comprising a task identifier, the conditions for executing the task and the task definition itself. (Lines beginning with a # character are treated as comments.) Tasks with long definitions may run onto additional lines and this can be signalled by using a special character such as "_" or "&", but can

nonetheless be thought of as occupying their own line. (In practice complex task definitions are best stored in a batch/script file and the batch/script called from the task file.)

In the present embodiment, each line begins with the numerical identifier, and in the example of Figure 2, the task identifiers are numbered 1 to 6.

The conditions for execution follow, with the next four portions representing the minute, hour, day of the month, and month respectively on which a task is to execute. In the present example, the day of the month and month are not specified for any tasks as the tasks execute at a frequency specified later in the line - and these unspecified portions are marked with a "*" character. For tasks which are only to execute once (per year), the day and month can be specified.

The next portion is used to specify task frequency, where numbers between 0 and 6 indicate the day of the week in which the task is to run. Job 1 runs every Sunday, represented by 6; whereas Job 2 runs every day of the week, represented by 0-6. Tasks to run on individual days are specified by separating the daily identifier with a comma. So for example, a task which is to run on Monday and Friday will be specified as 1,5. Alternatively, because the day of the week identifiers only comprise one digit, there may be no need to use a separator.

The next portion is used to indicate task dependency. Jobs 1 and 2 execute independently of other tasks and for these tasks this portion contains a "*" character. Jobs 3 to 6 are solely dependent on the completion of jobs 2,1,2 and 4 respectively, and in these cases no time/date has been specified. Nonetheless, it is possible to combine both a specification of time/date and a task dependency, so that, for example, a task can be specified to run no earlier than a given time/date, but only if another task has completed.

An almost infinite set of possible schedules can be specified using the above format, for example, by specifying 00 in the minute portion and 0-6 in the frequency portion, a task would be executed on the hour every day.

Finally, the task definition. The next portion defines the output file, if any, for a task, and in Figure 2, respective .OUT files are specified for each task. In the present embodiment, the final portion comprises the operating system command line for the task. These of course will be platform dependent commands which the scheduler is able to spawn. Thus, in spite of the scheduler itself being platform independent and

capable of running on any platform for which a suitable JVM is supplied, the scheduler can cause platform dependent tasks to be executed.

It is also possible to explicitly specify an error file for each task or alternatively a file having the same prefix as the output file and a .ERR suffix can be used.

Turning now to the read & execute thread 14. When the monitor thread 12 detects a change in the task file, it interrupts the thread 14. In response, the thread 14 reads in the changed task file 40 into a task list 18 in memory to replace the task list for the previous version of task file.

The thread notes the system time and date which are available to the JVM and then checks through the start time for each task in the task list 18. If the specified start time and date have been matched (or exceeded); the current day of the week is in the set specified for that task; and the task is not dependent on tasks that have yet to complete, the task is executed.

In the present embodiment, where dependent tasks can also be time dependent, this last dependency check is preferably recursive, as the execute thread must also check if those tasks on which a task is dependent are in turn dependent on others. Take, for example, Job 6. Even if Job 4 is not running (and so assumed to be completed), the thread must also check if Job 1 on which Job 4 is dependent is running. If this is complete then Job 6 can run.

If a task is to be executed, then a thread 16 is spawned for that task and the thread is passed the identifier, the output and error file name(s), if any, and the command line for the task. The thread passes the command line to the operating system where the command is executed and on completion, the thread closes..

The thread 14 detects the thread 16 closing and again parses through the task list 18 to determine if any tasks were waiting for this task to complete. If so, then so long as a later start time/date is not specified for such tasks, the task(s) is executed.

It should be seen that, in the present embodiment, where no minute, hour, day or month are specified for a task, as in the case of Jobs 3 to 6, then such tasks will only be executed in response to a task on which they are dependent being completed.

Maintaining a thread for each task being executed is a simple way for the scheduler to determine if dependent tasks are free to execute, as

it can very easily identify from the task identifiers associated with each thread 16, which tasks are being executed. It also means that as the scheduler continues to execute, even when the task file 40 changes, the scheduler can keep track of the tasks that are executing and so continue to operate correctly.

The output and error files specified for each task are kept for later analysis. In an alternative embodiment, however, the task file 40 can be extended to take into account the return codes provided by the operating system when a task completes. So, for example, Job 3 could be made dependent on Job 2 completing with a successful return code and Job 5 could be made dependent on Job 2 completing with any other return code, by including a return code portion in the line specifying the task in the task file 40.

It will be seen that the architecture of the job scheduler according to the invention facilitates its deployment within a heterogeneous distributed environment. In one embodiment operable in such an environment, a central application running on a manager's machine 60, Figure 1, creates and manages a number of task files 40, 40' and distributes them to the appropriate machines/platforms across a network.

Alternatively, the task file can be extended to include a machine identifier with each job identifier. This means that a central task file 50 can be updated by the manager and located where it is accessible to each scheduler 10 within the network. When the file 50 is updated, each scheduler 10 reads the file and creates either a task list 18, 18' of jobs that it is to execute, by selecting only these jobs specified to run on its machine or it can create a task list including all tasks and ignore those not designated to run on its machine. In any case, the central task file 50, as with the task files 40, 40', can be updated by the manager manually or by tasks, for example, Process A on Client #X or the test case running on the JVM under test on Client #1.

The ability of tasks to update the task file in any way by, for example, adding, amending or deleting tasks, allows for a great level of sophistication in the manner in which tasks can be scheduled. Take, for example, a task file which periodically checks a server to determine if an upgrade to software is required. The machine user may not want this to happen while using the machine and the network manager may not want this to happen on all machines at the same time to prevent network bottleneck. Thus, the task which checks for the upgrade can add or amend the task for carrying out the upgrade at a suitable time.

In a further alternative which overcomes any difficulty associated with accessing the task file 50 located on a different platform machine

to that of the scheduler, a single task file 40′ or 50 is placed by the manager on a machine accessible to a first scheduler - say the scheduler 10′ on Client #X. The first scheduler 10′ on identifying tasks to be executed on a second machine, say Client #1, communicates with the scheduler 10 on the second machine using, for example, remote method invocation (RMI), to indicate the task that is to be executed. On executing tasks, the other scheduler 10 spawns threads as before, so that before executing the remotely invoked task, it can check to determine if any tasks on which the remotely invoked task may be dependent are still executing. If not, the task is executed and, in any case, the task is placed on the scheduler's task list 18 which it monitors as before.

When tasks complete on the second scheduler's machine, the scheduler 10 notifies the first scheduler 10′, again by RMI, so that the first scheduler 10′ may cause further dependent tasks to execute on any machine in the network having a scheduler to which it can connect.

When the single task file 40′,50 is updated by the manager, the first scheduler 10′ notifies all other schedulers which in turn delete their task lists in memory and wait for further remotely invoked tasks as before. Nonetheless, this will not effect the fact that those schedulers continue to monitor currently executing tasks. Thus, it will be seen that the single task file 40′,50 acts as a seed for spawning tasks in a heterogeneous environment using only platform independent means.

It should be seen that for backup purposes, the second machines might write their task lists to a local task file. Nonetheless, there is logically a single central task file from which tasks emanate.

It should also be seen that while the monitoring, read and execute and task threads 12, 14 and 16 have been described as such, it is possible to implement the invention with this functionality integrated into a single thread or possibly even further divided into more threads.

Thus, in an alternative embodiment of the invention, the read and execute thread 14 spawns a thread 16 in response to the system time and date matching time and dates specified for a task in the task file. On completion of this task, the thread 16 checks the task list 18 for tasks that may be dependent on the task it has completed. It then spawns the further threads and closes, without the need for the read and execute thread 14 to examine the task list further.

In any case, it will be seen that the invention is not limited to Java per se and can be implemented in any platform independent language, for example, PERL which runs in an execution environment.

There are many possible applications of the scheduler of the invention above its simple application of running tasks at specific times. It will be seen that by placing a scheduler on each machine of a distributed network, it is possible via the task file to implement network management functionality at a highly abstracted level. Take, for example, an organisation who wish to monitor and maintain machines for Y2K compliance. As more information in relation to Y2K problems and fixes or patches become available, the task file can be updated accordingly to take into account such information.

Other variations of the invention falling within the scope of the claims will also be apparent. For example, improvements can be made to help prevent overloading a machine with Jobs/Tasks:

The scheduler can be limited to running a maximum number of Jobs/Tasks at any one time;
A job should not be started if there is an occurrence of the job already running; and
Jobs in a dependency chain (i.e. a series of jobs with each dependent upon on a prior job) should not overlap.

In the first case, the number of current running jobs is checked before starting a new job. If the maximum number are already running then the new jobs wait until completion of current jobs.

An example of the second case is where one long running Job (A) is scheduled to run each day. If this job take more than 24 hours then the next day's run may start before today's is finished. The second run of job (A) would then need to wait until today's run has finished.

In both the first and second cases, the scheduler needs to keep (in memory but perhaps with a backup to a file) a list of the jobs that are waiting/pending and to check the pending list each time a job finishes (after the dependent jobs). In the above example, when today's run of the job finishes, the second run could then start, as there would be no jobs running.

An example of the third case is where a series of jobs together take a long time. So if Jobs 1,2,3,4,5 are scheduled to run each day with Job 1 starting at 8am, the scheduler should not start Job 1 at 8am on the next day if Job 5 of the series from today's run is still running. Instead the Job 1 will be pended as above until all jobs in the series 1,2,3,4,5 have finished.

CLAIMS

1. A platform independent job scheduler operable in a platform dependent environment and cooperable with a task file comprising one or more platform dependent task definitions having associated conditions for execution, said scheduler comprising:

means for monitoring for a change to said task file;

means, responsive to said change, for reading said task file; and

means, responsive to the conditions for execution of any one of said one or more tasks being met, for executing said task.

2. A job scheduler as claimed in claim 1 wherein said conditions comprise a time and date and wherein said execution means is adapted to obtain a system time and date and, responsive to said system time and date matching any time and date specified for a task, execute said task.

3. A job scheduler as claimed in claim 2 wherein said conditions further include a specification of one or more days on which said task is to execute and wherein said execution means is responsive to said system time exceeding a time specified for a task and said system date falling on a day specified for execution of said task, to execute said task.

4. A job scheduler as claimed in claim 1 wherein each of said one or more task definitions includes an identifier of a machine on which said task is to execute and wherein said scheduler comprises means, responsive to said identifier indicating a remote machine on which said task is to execute, for communicating said task definition and its associated conditions to said remote machine.

5. A job scheduler as claimed in claim 4 wherein said conditions comprise a dependency of one task on completion of another and wherein said execution means is responsive to completion of said other task to execute said one task.

6. A job scheduler as claimed in claim 5 comprising means, responsive to a communication from said remote machine that said task has completed, for relaying said notification to said execution means.

7. A job scheduler as claimed in claim 5 wherein said dependency further comprises an operating system return code and wherein said execution means is responsive to an operating system code returned on completion of a task matching an operating system return code for a dependent task for executing said dependent task.

8. A job scheduler according to claim 1 wherein the platform dependent environment is a Java virtual machine and wherein the scheduler is a Java class.

9. A distributed heterogeneous system comprising a plurality of computer systems connected via a network, at least one of said computer systems having a platform different for the remainder of said computer systems and at least one of said computer systems including: a platform dependent environment operable on said computer system platform; a scheduler according to claim 1 operable on a respective platform dependent environment and a task file cooperable with said scheduler.

10. A distributed heterogeneous system according to claim 9 wherein at least one of said computer systems is a manager system from which task files are distributed to said at least one computer system including a scheduler.

11. A distributed heterogeneous system comprising a plurality of computer systems connected via a network, at least one of said computer systems having a platform different for the remainder of said computer systems, one of said computer systems including a task file and at least one of said computer systems including: a platform dependent environment operable on said computer system platform and; a scheduler according to claim 4 operable on a respective platform dependent environment.

12. In a platform independent job scheduler operable in a platform dependent environment and cooperable with a task file comprising one or more platform dependent task definitions having associated conditions for execution, a method of scheduling tasks comprising:

monitoring for a change to said task file;

responsive to said change, reading said task file; and

responsive to the conditions for execution of any one of said one or more tasks being met, executing said task.

13. A platform independent computer program product comprising computer program code stored on a computer readable storage medium for, when executed in a platform dependent environment operable on a computing device, scheduling jobs in accordance with a task file comprising one or more platform dependent task definitions having associated conditions for execution, the program code comprising:

means for monitoring for a change to said task file;

means, responsive to said change, for reading said task file; and

means, responsive to the conditions for execution of any one of said one or more tasks being met, for executing said task.

Application No:	GB 9924224.0	**Examiner:**	Mike Davis
Claims searched:	1-13	**Date of search:**	10 May 2000

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.R): G4A (AFN)

Int Cl (Ed.7): G06F

Other: Online: WPI, EPODOC, JAPIO

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
	None	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		